Speed and Adaptability of Overlap Fermion Algorithms

Rajiv V. Gavai* and Sourendu Gupta†

Department of Theoretical Physics, Tata Institute of Fundamental Research, Homi Bhabha Road, Mumbai 400005, India.

Robert Lacaze‡

Service de Physique Theorique, CEA Saclay, F-91191 Gif-sur-Yvette Cedex, France.

Abstract

We compare the efficiency of four different algorithms to compute the overlap Dirac operator, both for the speed, *i.e.*, time required to reach a desired numerical accuracy, and for the adaptability, *i.e.*, the scaling of speed with the condition number of the (square of the) Wilson Dirac operator. Although orthogonal polynomial expansions give good speeds at moderate condition number, they are highly non-adaptable. One of the rational function expansions, the Zolotarev approximation, is the fastest and is adaptable. The conjugate gradient approximation is adaptable, self-tuning, and nearly as fast as the ZA.

PACS numbers: 11.15.Ha, 12.38.Mh TIFR/TH/02-23, t02/091, hep-lat/0207005

*Electronic address: gavai@tifr.res.in
†Electronic address: sgupta@tifr.res.in
‡Electronic address: lacaze@spht.saclay.cea.fr

I. INTRODUCTION

A lack of consistent definition of chiral fermions on the lattice has hampered definitive and convincing investigations of chiral aspects of quantum chromodynamics (QCD) until now. Thus important physics issues, such as the spontaneous breaking of the chiral symmetry at low temperatures and its restoration at finite temperature, have remained hostages to technical questions such as the fine-tuning of the bare quark mass (Wilson fermions) or the precise number of massless flavours (staggered fermions). Recent developments in defining exact chiral symmetry on the lattice have therefore created exciting prospects of studying an enormous amount of physics in a cleaner manner from first principles. However, the corresponding Dirac operators are much more complicated. Without good control of the algorithms needed to deal with them, one is unlikely to derive the full benefit of their better chiral properties. Our goal in this paper is to evaluate the efficiency of the most widely used, or most promising, algorithms. By efficiency we mean both the speed of the algorithm, which is measured by the computer time required to achieve a certain accuracy in the solution, and the adaptability, which is measured by how the speed scales as the problem becomes harder. This study is made for various values of the required accuracy along with the corresponding analysis on the accuracy obtained for the expected properties of the resulting Dirac operator such as the Ginsparg-Wilson relation, the central circle relation, γ_5 hermiticity or normality. In particular, we have observed that these properties can be satisfied accurately only if the sign computation of the Wilson Dirac operator has high enough precision.

A. The overlap Dirac operator

One version of chiral fermions on the lattice is the overlap formalism. The overlap Dirac operator (D) is defined [1] in terms of the Wilson-Dirac operator (D_w) by the relation

$$D = 1 + D_w (D_w^\dagger D_w)^{-1/2}. \tag{1}$$

In this paper we shall use the shorthand notation

$$M = D_w^\dagger D_w \ . \tag{2}$$

The Wilson-Dirac operator D_w (for lattice spacing $a = 1$) is given by

$$D_w = \frac{1}{2} \sum_\mu [\gamma_\mu (\partial_\mu + \partial_\mu^*) - \partial_\mu \partial_\mu^*] + m \ , \tag{3}$$

where ∂_μ and ∂^*_μ are (gauge covariant) forward and backward difference operators respectively. It has been shown that as long as the mass m is in the range $-2 < m < 0$, the above overlap Dirac operator is well-defined, and corresponds to a single massless fermion. Furthermore, it satisfies the Ginsparg-Wilson relation [2]

$$\gamma_5 D + D\gamma_5 = D\gamma_5 D, \tag{4}$$

which leads to a good definition of chirality on the lattice and has been shown to correspond to an exact chiral symmetry on the lattice.

The overlap Dirac operator, D, enjoys many nice properties in addition to the Ginsparg-Wilson relation in Eq. (4). In particular, it satisfies γ_5-Hermiticity—

$$D^\dagger = \gamma_5 D \gamma_5. \tag{5}$$

Together with the Ginsparg-Wilson relation, this implies that D is normal, *i.e.*,

$$[D, D^\dagger] = 0. \tag{6}$$

Normality clearly means that D and $D^\dagger$ have the same eigenvectors. Eqs. (4,5) also imply

$$D + D^\dagger = D^\dagger D, \tag{7}$$

and hence the eigenvalues of D and $D^\dagger$ lie on the unit circle centered at unity on the real line, implying that $D-1$ is unitary. We define measures of numerical errors on each of these quantities, and relations between them in Section II.

B. Numerical algorithms

All computations of hadronic correlators involve the determination of the fermion propagator D^{-1}, and need a nested series of two matrix iterations for their evaluation, since each step in the numerical inversion of D involves the evaluation of $M^{-1/2}$. This squaring of effort makes a study of QCD with overlap quarks very expensive.

This problem defines for us the properties that an efficient algorithm to deal with $M^{-1/2}$ must have. First, it should achieve a given desired accuracy as quickly as possible. The need for accuracy is clear: the accuracy to which the Ginsparg-Wilson relation in Eq. (4) is satisfied depends on the accuracy achieved in the computation of $M^{-1/2}$. The second, and

equally important, requirement is that the method should adapt itself easily to matrices with widely different condition numbers—

$$\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}, \tag{8}$$

where $\lambda_{\min}$ and $\lambda_{\max}$ are, respectively, the minimum and the maximum eigenvalues of M. Adaptability is needed because in QCD applications the eigenvalue spectrum of M can fluctuate over many orders of magnitude from one configuration to the next. Since the condition number on a configuration is *a priori* unknown, a method with low adaptability will end up either being inefficient or inaccurate or even both. Algorithms, which are adaptable in principle, may require tuning of parameters by hand, or they may incorporate a procedure for self-tuning. Clearly, self-tuning algorithms are the ones that can best deal with fluctuating condition numbers in any real situation.

In this paper we examine the speed and adaptability of several different algorithms to compute the inverse square root, $M^{-1/2}$, of Hermitean matrices (in our applications the eigenvalues of M are non-negative) acting on a vector. Several algorithms for this have been proposed in the literature.

We do not consider the first algorithm to be proposed, since this requires a matrix inversion to be performed at each step of an iteration to determine $M^{-1/2}$ [3]. Later algorithms are more efficient. These fall into two classes— expansions of $1/\sqrt{z}$ in appropriate classes of functions (rational functions [4] or orthogonal polynomials [5]), and iterative methods [6]. We have analyzed four derived algorithms, namely the Optimized Rational Approximation (ORA) [7], the Zolotarev Approximation (ZA, which is also a rational expansion) [8, 9], the Chebychev Approximation (CA, a polynomial expansion) [10] and the Conjugate Gradient Approximation (CGA, an iterative method) [11]. We find that an expansion in Chebychev polynomials is the fastest when κ is moderately large, but it suffers from various instabilities including a lack of adaptability. Rational expansions cure many of the instabilities of polynomial expansions; indeed the ZA is the fastest and is adaptable but not self-tuning. An iterative method is the only fully self-tuned algorithm, and it turns out to be reasonable also from the point of view of speed.

C. Algorithmic costs and adaptability

We make two different estimates of the cost of each algorithm. The complexity, $\mathcal{C}$, counts the number of arithmetic operations required to achieve the solution to the problem and is a measure of speed independent of the specific machine on which the algorithm is implemented. The spatial complexity, $\mathcal{S}$, is the memory requirement for the problem. While timing runs on particular machines on chosen test configurations are instructive, the scaling of speed for each algorithm with physical and algorithmic parameters is provided by our estimates of $\mathcal{C}$.

Our estimate of the adaptability, $\mathcal{A}$, of each algorithm is the following. If the scalar algorithm for $1/\sqrt{z}$ is tuned to have maximum relative error ε in the range $[z_{\min}, z_{\max}]$, then we find the smallest range $[z'_{\min}, z'_{\max}]$ where the error is at most 10ε. Note that the second interval cannot be smaller than the first. In terms of these quantities we define

$$\mathcal{A} = \frac{\log(z'_{\max}/z'_{\min})}{\log(z_{\max}/z_{\min})} - 1. \tag{9}$$

The least adaptable algorithms have small values of $\mathcal{A}$ ($\mathcal{A} > 0$). $\mathcal{A}$ is a measure of the relative accuracy achieved in a fixed CPU time for the same algorithm running on two different configurations with condition numbers $\kappa = z_{\max}/z_{\min}$ and $\kappa' = z'_{\max}/z'_{\min}$. In conjunction with estimates of $\mathcal{C}$, it also contains information about the scaling of CPU time required to achieve the same accuracy on the two configurations.

D. Numerical tests

Our numerical tests were performed with three typical $SU(3)$ gauge configurations on a 4×12^3 lattice at $\beta = 5.80$ (*i.e.*, $T = 1.25T_c$). The configuration A had eigenvalues of M in the range $[0.032, 32]$ so that $\kappa = 10^3$. The configuration B had eigenvalues in the range $[7.2 \times 10^{-5}, 32]$, giving $\kappa = 4.4 \times 10^5$. Finally, configuration C had eigenvalues in the range $[8.9 \times 10^{-9}, 32]$ and hence $\kappa = 3.6 \times 10^9$. Configuration A is one of the easiest configuration we found in our simulations, and there were several configurations with κ of order 10^5–10^9 in the data set we worked with in [11]. If there is a single scale in the level spacing of the eigenvalues of M, then we expect $\kappa = \mathcal{O}(V) \approx 7 \times 10^3$ on our test configurations. We conclude that A is indeed a little easier than the generic configuration and B and C are successively harder. The CPU times we quote in our tables are obtained on a Fujitsu

VPP5000, which is a vector computer. Our computations ran on this machine at a speed of around 4.1 Gigaflops.

In sections III–VI we describe and analyze the four algorithms and also present data on precision and time measurements on these three test configurations. In this work we have not investigated the performance of the algorithms with deflation (explicit subtraction) of some eigenvectors. However, we do remark on the precision required for deflation and the propagation of errors due to such a subtraction. Section VII contains a comparison of the numerical results and our conclusions.

II. ERRORS

In general, every numerical method to compute $M^{-1/2}$ constructs an operator L which applied to a vector Φ gives the vector

$$X = L[\Phi] \qquad \text{with} \qquad L[\Phi] = M^{-1/2}\Phi + E[\Phi], \tag{10}$$

where E is the error in the approximation, L, to the matrix $M^{-1/2}$. Typically, these operators L and E are not matrices because the algorithms can introduce a dependence on Φ which is not linear. The error $E[\Phi]$ on the computation of $M^{-1/2}$ leads to the violation of the properties in Eqs. (4-7). In our numerical tests we have investigated five measures of the accuracy of the algorithms through norms of the following operators—

$$\begin{aligned} Z_{1/2} &= ML^2 - 1, \quad Z_{GW} = D\gamma_5 + \gamma_5 D - D\gamma_5 D, \quad Z_N = DD^\dagger - D^\dagger D, \\ Z_H &= D^\dagger - \gamma_5 D \gamma_5, \qquad Z_{CC} = D + D^\dagger - D^\dagger D, \end{aligned} \tag{11}$$

where $D = 1 + D_w L$ and $D^\dagger = 1 + L D_w^\dagger$. Each of these operators is zero when $E[\Phi] = 0$. With Gaussian random vectors Φ, we have measured the deviations from this exact value through

$$\epsilon_i = |Z_i\Phi|/|\Phi| \qquad \text{and} \qquad \epsilon_i' = \Phi^\dagger Z_i \Phi/|\Phi|^2. \tag{12}$$

Here, and later, we use the notation $|v| = \sqrt{v^\dagger v}$ for any complex vector v. Note that ϵ_i are real and non-negative whereas ϵ_i' are complex in general.

A. Polynomial approximation

The polynomial approximation consists of writing

$$L = \sum_{i=1}^{N_O} b_i M^i, \tag{13}$$

where b_i are constants. It is clear that both L and E are matrices in this case and

$$[L, M] = 0. \tag{14}$$

Since in numerical implementations of D_w, γ_5-Hermiticity is accurate to machine precision, *i.e.*, $|(D_w^\dagger - \gamma_5 D_w \gamma_5)\Phi| = 0$, one has the following relation:

$$\gamma_5 D_w M^n \gamma_5 = M^n D_w^\dagger. \tag{15}$$

Its direct consequence is

$$Z_H = L D_w^\dagger - \gamma_5 D_w L \gamma_5 = 0. \tag{16}$$

Using Eqs.(14-15), one can easily obtain the following relations between the various Z's :

$$\begin{aligned} Z_{CC} &= -Z_{1/2}, \\ Z_{GW} &= \gamma_5 Z_{CC}, \\ Z_N &= Z_{CC} - \gamma_5 Z_{CC} \gamma_5. \end{aligned} \tag{17}$$

As a consequence,

$$\begin{aligned} |\epsilon'_{1/2}| &= |\epsilon'_{CC}|, \\ \epsilon_{1/2} &= \epsilon_{CC} = \epsilon_{GW}, \\ \epsilon'_H &= \epsilon_H = 0. \end{aligned} \tag{18}$$

Expanding $Z_{1/2} = ML^2 - 1 = Z_{1/2}^\dagger$ in powers of E and retaining only the leading order terms, we find that $Z_{1/2}^\dagger Z_{1/2} = 4E^2 M$ and $Z_{1/2} = 2EM^{1/2}$. Defining the averages of ϵ_i and ϵ'_i over an ensemble of Φ as $\overline{\epsilon_i^2} = \mathrm{Tr}\, Z_i^\dagger Z_i$ and $\overline{\epsilon'_i} = \mathrm{Tr}\, Z_i$, one obtains,

$$\begin{aligned} \overline{\epsilon_{1/2}^2} = \overline{\epsilon_{CC}^2} = \overline{\epsilon_{GW}^2} &= 4 \int d\lambda \rho(\lambda) \lambda \epsilon^2(\lambda), \\ \overline{\epsilon'_{1/2}} = -\overline{\epsilon'_{CC}} &= 2 \int d\lambda \rho(\lambda) \sqrt{\lambda} \epsilon(\lambda), \\ \overline{\epsilon'_{GW}} &= 2 \int d\lambda \Delta\rho(\lambda) \sqrt{\lambda} \epsilon(\lambda), \end{aligned} \tag{19}$$

where $\rho(\lambda)$ is the density of eigenvalues of M, $\epsilon(\lambda)$ is the error in the approximation and $\Delta\rho(\lambda) = \rho_+(\lambda) - \rho_-(\lambda)$, the difference between the spectral densities (of M) in the chiral positive and negative sectors. Note that $\sqrt{\lambda}\epsilon(\lambda)$ is the relative error in the determination of the inverse square root, and all the integrals depend only on this relative error. Since there is no reason for $\rho(0)$ to vanish, it is clear that the error in the expansion must remain under control even as $\lambda \to 0$. Clearly, this is impossible to arrange in polynomial expansions for $1/\sqrt{\lambda}$. However, a finite sample of gauge configurations does not need full control over $\epsilon(0)$, but only for $\epsilon(\lambda_<)$, where $\lambda_<$ is the smallest eigenvalue encountered in the sample. To achieve this while optimizing CPU costs on configurations where all the eigenvalues are much larger requires the algorithm to be adaptive.

It was assumed above that no deflation has been performed, or that deflation has been performed with no arithmetic errors. We comment on the effects of deflation in Section VII.

B. Rational approximation

In case of a rational function approximation to $M^{-1/2}$, one writes the operator L as

$$L = \sum_{i=1}^{N_O} \frac{b_i}{M + d_i} \tag{20}$$

E here depends on the order N_O and the accuracy of the inversion of $(M + d_i)$. If the inversion can be achieved with infinite precision, then L is a matrix again which commutes with M and the analysis of the previous subsection applies in full. If, on the other hand, the error due to the inversion dominates, then for many algorithms, such as the Conjugate Gradient, L depends explicitly on the vector Φ in a complicated way and it is not a matrix. One has to compute the different errors explicitly and study their behavior as in iterative methods. Thus the behavior of errors from rational approximation case interpolates between that of the polynomial approximation and an iterative method according to the relation between the order and the precision of the inversion.

III. FIXED ORDER: CHEBYCHEV APPROXIMATION

The first use of the polynomial approximation utilized Legendre polynomials [5]. Later the same group proposed a more robust version using the Chebychev approximation (CA)

[10]. As is well known, when expanding any function, $f(z)$ in a fixed range $z_{\min} \leq z \leq z_{\max}$, to a given order N_O through orthogonal polynomials, the use of Chebychev polynomials minimizes the maximum error on the function to be approximated.

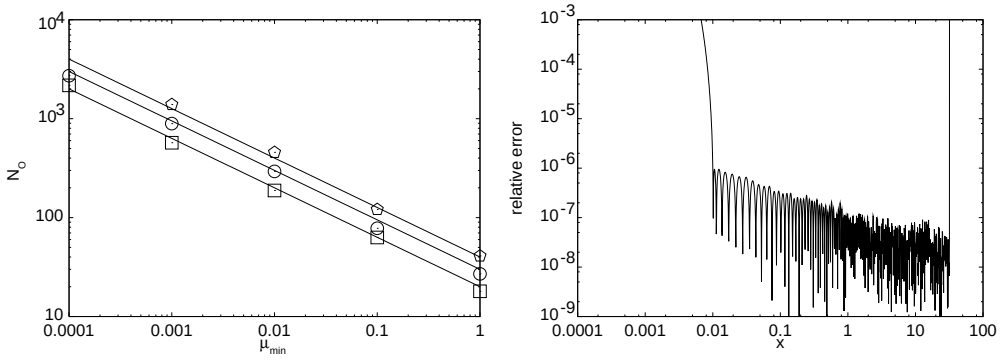

FIG. 1: The panel on the left shows the order of Chebychev expansion, N_O required to reach an accuracy of 10^{-3} (boxes), 10^{-4} (circles) and 10^{-6} (pentagons) in the range $[\mu_{\min}, 32]$ under variation of $\mu_{\min}$ (the lines are proportional to $\log(1/\varepsilon)/\sqrt{\mu}_{\min}$). The panel on the right shows the error for a Chebychev expansion in the range $[0.01, 32]$ with $N_O = 400$.

For the function $1/\sqrt{z}$, the coefficients of the Chebychev expansion for $z_{\min} \leq z \leq z_{\max}$ to order N_O are

$$C_k = \frac{2\sqrt{2}}{N_O} \sum_{j=1}^{N_O} \frac{\cos((k-1)(j-\frac{1}{2})\pi/N_O)}{[z_{\min} + z_{\max} + (z_{\min} - z_{\max})\cos((j-\frac{1}{2})\pi/N_O)]^{1/2}}. \tag{21}$$

Applying this approximation to a matrix M corresponds to finding $L\Phi$ (see Eq. 10) by the expansion—

$$L\Phi = \frac{1}{2} c_1 \Phi^{(0)} + \sum_{k=2}^{N_O} c_k \Phi^{(k-1)}, \tag{22}$$

where the successive vectors $\Phi^{(n)}$ can be found by the iteration $\Phi^{(0)} = \Phi$ and

$$\Phi^{(n)} = \frac{2}{z_{\max} - z_{\min}} \left[2M\Phi^{(n-1)} - (z_{\max} + z_{\min})\Phi^{(n-1)} \right] - \Phi^{(n-2)}, \tag{23}$$

for $n \geq 2$. For $n = 1$ the term $\Phi^{(-1)}$ is dropped from the recursion.

There are various sources of error, which we now analyze in succession. For the scalar version of the algorithm (or for each eigenvector of M separately), with fixed $z_{\min}$, $z_{\max}$ and parameter N_O, there is an error in the approximation of $1/\sqrt{z}$ which we call $\epsilon(z)$. For computations at arbitrary precision with a fixed range, $[z_{\min}, z_{\max}]$, $\epsilon(z)$ depends entirely on

N_O	$\epsilon_{1/2}$	ϵ_{GW}	ϵ_{CC}	time
30	0.273×10^{-1}	0.273×10^{-1}	0.273×10^{-1}	0.3
65	0.174×10^{-2}	0.174×10^{-2}	0.174×10^{-2}	0.7
100	0.136×10^{-3}	0.136×10^{-3}	0.136×10^{-3}	1.1
135	0.113×10^{-4}	0.113×10^{-4}	0.113×10^{-4}	1.5
165	0.140×10^{-5}	0.140×10^{-5}	0.140×10^{-5}	1.8
200	0.125×10^{-6}	0.125×10^{-6}	0.125×10^{-6}	2.2
235	0.113×10^{-7}	0.113×10^{-7}	0.113×10^{-7}	2.5
270	0.102×10^{-8}	0.102×10^{-8}	0.102×10^{-8}	2.9
300	0.134×10^{-9}	0.134×10^{-9}	0.134×10^{-9}	3.4
330	0.174×10^{-10}	0.174×10^{-10}	0.174×10^{-10}	3.7
360	0.230×10^{-11}	0.230×10^{-11}	0.230×10^{-11}	4.1
390	0.391×10^{-12}	0.391×10^{-12}	0.391×10^{-12}	4.4
420	0.210×10^{-12}	0.209×10^{-12}	0.209×10^{-12}	4.8
450	0.226×10^{-12}	0.226×10^{-12}	0.226×10^{-12}	4.9

TABLE I: Runs with the CA adjusted to the interval $[0.032, 32]$ for varying N_O on the configuration A. The last column gives the CPU seconds used on a Fujitsu VPP5000.

N_O	$\epsilon_{1/2}$	ϵ_{GW}	ϵ_{CC}	time
100	0.236×10^{-1}	0.236×10^{-1}	0.236×10^{-1}	1.1
500	0.294×10^{-2}	0.294×10^{-2}	0.294×10^{-2}	5.4
1000	0.485×10^{-3}	0.485×10^{-3}	0.485×10^{-3}	10.8
1500	0.108×10^{-3}	0.108×10^{-3}	0.108×10^{-3}	16.9
2000	0.292×10^{-4}	0.292×10^{-4}	0.292×10^{-4}	21.8
3000	0.254×10^{-5}	0.254×10^{-5}	0.254×10^{-5}	32.7
4000	0.267×10^{-6}	0.267×10^{-6}	0.267×10^{-6}	43.7

TABLE II: Runs with the CA adjusted to the interval $[7.2 \times 10^{-5}, 32]$ on the configuration B. The last column shows the CPU seconds used on a Fujitsu VPP5000.

N_O	$\epsilon_{1/2}$	ϵ_{GW}	ϵ_{CC}	time
100	0.253×10^{-1}	0.253×10^{-1}	0.253×10^{-1}	1.1
500	0.788×10^{-2}	0.788×10^{-2}	0.788×10^{-2}	5.5
1000	0.471×10^{-2}	0.471×10^{-2}	0.471×10^{-2}	10.9
1500	0.395×10^{-2}	0.395×10^{-2}	0.395×10^{-2}	16.2
3000	0.460×10^{-2}	0.460×10^{-2}	0.460×10^{-2}	32.4

TABLE III: Runs with the CA adjusted to the interval $[8.9 \times 10^{-9}, 32]$ on the configuration C. The last column shows the CPU seconds used on a Fujitsu VPP5000.

N_O. To keep the absolute relative error bounded, $|\epsilon(z)|\sqrt{z} \leq \varepsilon$, as $z_{\min}$ changes, we must tune

$$N_O \propto \log\left(\frac{1}{\varepsilon}\right)\sqrt{\frac{1}{z_{\min}}} \simeq \log\left(\frac{1}{\varepsilon}\right)\sqrt{\kappa}, \tag{24}$$

as shown in Figure 1. The last expression follows if we choose $z_{\min} = \lambda_{\min}$ and $z_{\max} = \lambda_{\max}$.

However, as shown in the right panel of the figure, tuning N_O in this manner causes the relative error outside the range to blow up. The CA has no tolerance to violations of the requirement on the range. This is easy to understand. In any polynomial approximation, with decreasing $z_{\min}$ larger values of N_O are required for keeping the error fixed within the interval $[z_{\min}, z_{\max}]$. However, outside this interval, the error then increases as

$$\epsilon(z) \propto \left(\frac{2z - z_{\min} - z_{\max}}{z_{\max} - z_{\min}}\right)^{N_O}, \qquad \text{for } z < z_{\min} \text{ or } z > z_{\max}, \tag{25}$$

and hence the error increases faster as $z_{\min}$ decreases. Solving this for z, given some fixed value of $\epsilon(z)/\varepsilon$, we can easily see that for large κ and fixed precision ε,

$$\mathcal{A} \propto \exp(-\alpha\sqrt{\kappa}) \tag{26}$$

where α is some number.

The effect of finite arithmetic precision can also be analyzed easily since the iteration in Eq. (23) is linear. Any arithmetic error, $\delta^{(m)}$, in $\Phi^{(m)}$ of the order of the machine precision remains in control whenever all eigenvalues, λ of M satisfy $z_{\min} \leq \lambda \leq z_{\max}$. If any eigenvalue of M lies outside this range, then the iteration magnifies the error geometrically—

$$\delta^{(m+n)} \simeq \left(\frac{2\lambda_{\min} - z_{\min} - z_{\max}}{z_{\max} - z_{\min}}\right)^n \delta^{(m)}, \tag{27}$$

This is the vector version of the low adaptability of this algorithm. If estimates of $\lambda_{\min}$ and $\lambda_{\max}$ for M are available, then, in view of this instability, it is best to choose $z_{\min} < \lambda_{\min}$ and $z_{\max} > \lambda_{\max}$.

The complexity of this algorithm is clearly dominated by the time required to operate upon a vector by the matrix M in the iterations in Eq. (23). For the Wilson-Dirac matrix this time is of order V. Neglecting the time taken for scalar operations in the remainder of the algorithm, and also the order V time for vector additions, in comparison with this, we have—

$$\mathcal{C}_{CA} \simeq wN_O V \simeq w'V \log\left(\frac{1}{\varepsilon}\right)\sqrt{\kappa}, \tag{28}$$

where wV is the complexity of operating upon a vector by M and w, w' are constants. Apart from the gauge configuration, in QCD applications the storage required is for the three vectors needed for the iteration in Eq. (23). The space complexity is therefore

$$\mathcal{S}_{CA} = 8N_c(N_c+3)V, \tag{29}$$

(for N_c colors) neglecting storage for scalars.

With a fixed N_O, the precision of the algorithm deteriorates sharply when one or more of the eigenvalues of M lie outside the interval $[z_{\min}, z_{\max}]$, as shown in Figure 1 and by the low value of $\mathcal{A}$ in Eq. (26). This is often sought to be corrected for by deflating, *i.e.*, explicitly dealing with the eigenspace of the lowest eigenmodes, and applying the algorithm to the orthogonal space. This would keep the accuracy constant as the condition number changes. If N_D vectors need to be deflated, then the contribution to $\mathcal{C}$ clearly increases as $(N_D V)^2$, since each vector has to be orthogonalized with respect to every other. Also, $\mathcal{S}$ increases as $N_D V$ due to the necessity of storing the vectors. In order to achieve a target precision, ε, on all gauge configurations, we are forced to deflate all vectors with $\lambda < z_{\min}$. Working with a fixed N_D forces us to do unnecessarily large amount of work on most configurations, while still failing on a small set of configurations. As a result, N_D has to be chosen appropriately for each configuration. With deflation then we have

$$\begin{aligned} \mathcal{C}_{CA} &= w'V \log\left(\frac{1}{\varepsilon}\right)\sqrt{\kappa_{eff}} + w''V^2\langle {N_D}^2\rangle, \\ \mathcal{S}_{CA} &= 8N_c(N_c+3)V + 8N_c\langle N_D\rangle V, \end{aligned} \tag{30}$$

where the angular brackets denote averages over gauge configurations, w'' is a constant independent of V and N_D, and κ_{eff} is the condition number of the matrix after deflating

N_D vectors. With careful programming we can arrange to make $w'' < w'$, although they cannot differ by orders of magnitude (w'' can depend on ε and $\lambda_{\min}$).

We have not investigated the expectation values of N_D. However, when we change the volume at fixed physics, we expect that $\langle N_D \rangle \propto V\overline{\rho}$, where $\overline{\rho}$ is the average density of eigenvalues of M near $\lambda_{\min}$. Since deflation is designed to hold κ_{eff} fixed, this means that for large enough V the complexity $\mathcal{C}_{CA} \propto V^4$. On the other hand, κ should generically grow linearly in V. Hence, on sufficiently large volumes, without deflation we would have $\mathcal{C}_{CA} \propto V^{3/2}$. For best actual performance, one would have to tune the playoff between these two limits.

A further complication arises in CA, and indeed, in any method which utilizes a polynomial expansion. For any fixed finite precision, the deflation of $\Phi^{(0)}$ is inaccurate since each component of the deflated vector is in error at least in the least significant bit. Since the CA iteration of Eq. (23) is not stable on the deflated eigenspace, this error blows up geometrically as in Eq. (27). Consequently, more and more bits are corrupted, as the iteration proceeds, and the process may eventually render the whole computation unusable. Note that this problem becomes more acute with decreasing κ, even if several eigenvectors are deflated. To prevent the error from swamping the result in this fashion, one has to reorthogonalize $\Phi^{(n)}$ repeatedly (this process itself is not free of complications, see [12]). This involves finding N_D dot products and subtracting N_D vectors. While this is crucial in maintaining the accuracy of the result, it does not change the complexity, and we still have the results in Eq. (30).

A different approach, and the one we have adopted for our tests, is to start the algorithm by making an estimate of $\lambda_{\min}$ and $\lambda_{\max}$ and then to select N_O accordingly. This obviates any need for deflation, and controlling the rounding errors in such a method. The algorithm is well-behaved, both with respect to precision and propagation of rounding errors, whenever $z_{\min} \leq \lambda_{\min} \leq \lambda_{\max} \leq z_{\max}$. However, in this case the complexity rises as $\sqrt{\kappa}$.

As shown in Table I, the algorithm performs well on configuration A. N_O needed to achieve a given value of $\epsilon_{1/2}$ is seen to rise logarithmically, as argued above. Consequently, $\epsilon_{1/2}$ can be made very small and the required chiral properties obtained at any desired precision. Note also that the equalities (18) are exactly satisfied, as expected, and remain so for configurations B and C too, as seen from the Tables II and III respectively. However, the algorithm was found to be extremely slow for these configurations and saturated at

$N_O \sim 1500$ with $\epsilon_{1/2} = 4 \times 10^{-3}$ on the configuration C, leading to the same precision for both the GW relation and the unit circle property of D.

IV. FIXED ORDER: OPTIMIZED RATIONAL APPROXIMATION

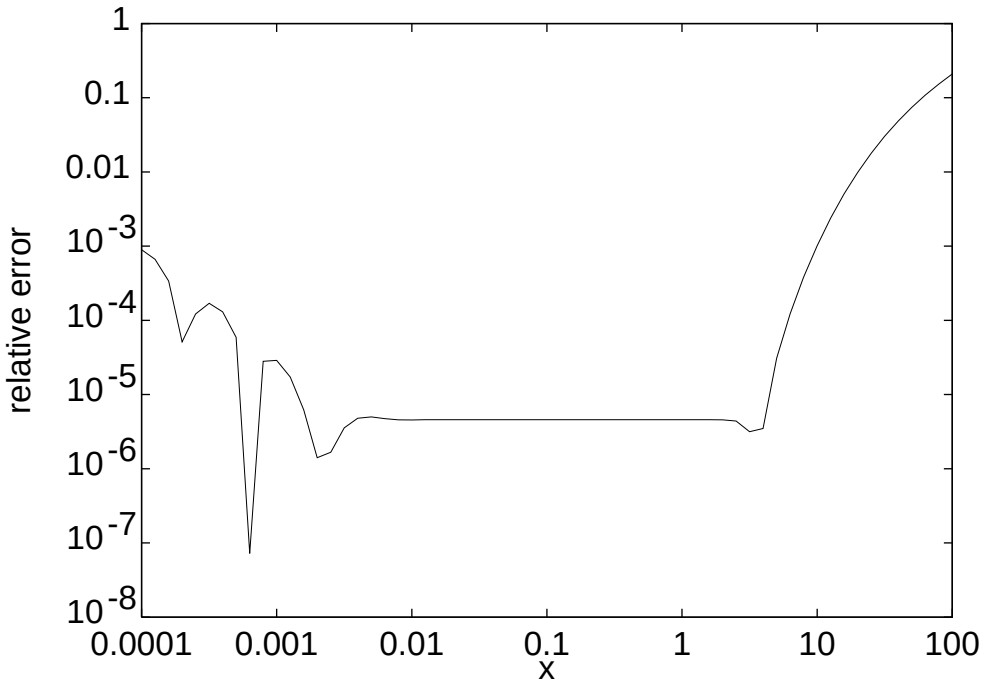

FIG. 2: The relative error, $|\epsilon(z)|\sqrt{z}$, in the ORA with $N_O = 14$ fitted to the range $[0.01, 1]$ for computation in and outside the range.

The first algorithm to compute $M^{-1/2}$ through a rational expansion was a polar formula introduced by Neuberger [4]. An improved version [7], called the optimized rational approximation, uses coefficients obtained numerically to give the approximation

$$L\Phi = \sqrt{\alpha} \left[c_0 + \sum_{k=1}^{N_O} \frac{c_k}{\alpha M + d_k} \right] \Phi, \tag{31}$$

where α is an arbitrary scale whose choice we describe later, and the values of c_k and d_k for a given N_O are obtained by optimizing the fit on some fixed interval $[z_{\min}, z_{\max}]$ using the Remez algorithm (see [7] for details). The inversion of $(\alpha M + d_k)$ is made by a multimass conjugate gradient stopped according to a value ϵ_{CG} for the residual. This version is used in particular in [13], [14], and [15].

α	$10^3\epsilon_{1/2}$	$10^3\epsilon_{GW}$	$10^3\epsilon_{1/2}$	$10^3\epsilon_{GW}$	$10^3\epsilon_{1/2}$	$10^3\epsilon_{GW}$
1.00	11.2	11.3	11.2	11.2	11.2	11.2
0.50	1.74	2.01	1.44	1.44	1.44	1.44
0.20	0.987	1.40	0.0780	0.116	0.0257	0.0273
0.15	0.986	1.40	0.0744	0.114	0.0112	0.0143
0.12	0.986	1.40	0.0744	0.114	0.0110	0.0142
0.10	0.986	1.40	0.0744	0.114	0.0111	0.0142
0.08	0.986	1.40	0.0744	0.114	0.0111	0.0143
0.05	0.986	1.40	0.0744	0.114	0.0111	0.0142

TABLE IV: Tuning α in ORA for a fixed configuration with three different values of ϵ_{CG}. Since this fixes the upper part of the range of eigenvalues, we expect little change from one configuration to another, and $\alpha = 0.1$ is a global choice.

ϵ_{CG}	N_{CG}	$\epsilon_{1/2}$	ϵ_{GW}	ϵ_{CC}	time
10^{-1}	25	0.179×10^{-1}	0.257×10^{-1}	0.539×10^{-1}	0.5
10^{-2}	63	0.986×10^{-3}	0.140×10^{-2}	0.614×10^{-2}	1.1
10^{-3}	99	0.744×10^{-4}	0.114×10^{-3}	0.507×10^{-3}	1.6
10^{-4}	134	0.111×10^{-4}	0.142×10^{-4}	0.397×10^{-4}	2.1
10^{-5}	166	0.916×10^{-5}	0.919×10^{-5}	0.966×10^{-5}	2.6
10^{-6}	198	0.914×10^{-5}	0.914×10^{-5}	0.914×10^{-5}	3.1

TABLE V: Runs with the ORA for $N_O = 14$ optimized in the interval $[0.01, 1]$ and $\alpha = 0.1$ on configuration A. The last column gives the CPU seconds used on a Fujitsu VPP5000.

Taking the values of c_k and d_k for $N_O = 14$, $z_{\min} = 0.01$ and $z_{\max} = 1$ from [7], we show in Figure 2 the relative accuracy, $|\epsilon(z)|\sqrt{z}$ for the expansion both inside and outside the fitted range. It is clear from the figure that the algorithm rapidly degrades outside the chosen range. A numerical computation shows that

$$\mathcal{A} \approx 1, \tag{32}$$

so that it has much higher adaptability than the CA. Nevertheless, it makes large errors on

ϵ_{CG}	N_{CG}	$\epsilon_{1/2}$	ϵ_{GW}	ϵ_{CC}	time
10^{-1}	71	0.148×10^{-1}	0.243×10^{-1}	0.542×10^{-1}	1.2
10^{-2}	332	0.767×10^{-4}	0.113×10^{-3}	0.799×10^{-2}	5.0
10^{-3}	363	0.220×10^{-4}	0.229×10^{-4}	0.499×10^{-2}	5.5
10^{-4}	395	0.208×10^{-4}	0.208×10^{-4}	0.160×10^{-2}	6.0
10^{-5}	426	0.208×10^{-4}	0.208×10^{-4}	0.209×10^{-4}	6.4

TABLE VI: Runs with the ORA for $N_O = 14$ optimized in the interval $[0.01, 1]$ and $\alpha = 0.1$ on configuration B. The last column shows the CPU seconds used on a Fujitsu VPP5000.

ϵ_{CG}	N_{CG}	$\epsilon_{1/2}$	ϵ_{GW}	ϵ_{CC}	time
10^{-1}	317	0.161×10^{-1}	0.246×10^{-1}	0.547×10^{-1}	4.9
10^{-2}	781	0.333×10^{-2}	0.333×10^{-2}	0.111×10^{-1}	11.9
10^{-3}	815	0.333×10^{-2}	0.333×10^{-2}	0.771×10^{-2}	12.5

TABLE VII: Runs with the ORA for $N_O = 14$ optimized in the interval $[0.01, 1]$ and $\alpha = 0.1$ on configuration C. The last column shows the CPU seconds used on a Fujitsu VPP5000.

the eigenvalues of M which are greater than $z_{\max}$. It is easy to see that many eigenvalues are greater than unity, and a scaling factor α is therefore needed to bring these into range. The tuning of α is shown in Table IV. For the conjugate gradient inversion of each term, when ϵ_{CG} is large, it determines the accuracy of the solution. Hence ϵ_{CG} must be kept small enough so that the accuracy is ϵ.

In order to specify the scaling of the CPU time in each algorithm with various parameters of the problem, we count the complexity of the method. For this algorithm, $\mathcal{C}$ is proportional to the number of steps of the Conjugate Gradient inversion, N_{CG}. It can be proved that N_{CG} grows no faster than $\sqrt{\kappa}$. However, from the observed convergence rate of the CG iterations (shown in Section VII), we see that $N_{CG} \propto \log(1/\epsilon_{CG}) \log \kappa$. This expression can be used when ϵ_{CG} is tuned to be smaller than the error shown in Figure 2. For each step of the multimass CG inversion, the complexity is dominated by the time required to operate upon a vector by the matrix M. For the Wilson-Dirac matrix this is of order V. Neglecting the

time taken for scalar operations and also the order V time for vector additions, we have—

$$\mathcal{C}_{ORA} \simeq wN_{CG}V \simeq w'V \log\left(\frac{1}{\epsilon_{CG}}\right) \log\kappa, \tag{33}$$

where wV is the complexity of operating upon a vector by M, and w' is a constant. The dependence of N_{CG} on V is very weak for realistic N_{CG} and is neglected. The memory requirement is essentially for the storage of the gauge configuration and for $2+2N_O$ vectors required to build up the approximation. The space complexity is therefore

$$\mathcal{S}_{ORA} = 8N_c(N_c+2+2N_O)V, \tag{34}$$

(for N_c colors) neglecting storage for scalars.

With a fixed value of N_O, the precision of the algorithm deteriorates when the eigenvalues of M lie outside the range $[z_{\min}, z_{\max}]$, as shown in Figure 2. For D_w, the highest eigenvalue remains in the vicinity of 32 for most configurations, whereas the lowest eigenvalue may fluctuate by several orders of magnitude. The large eigenvalues are brought into range by tuning $\alpha < 1$ as shown already. However, this drives the lowest eigenvalues further out of the range, thus degrading performance. As a result, it is necessary to deflate, *i.e.*, explicitly deal with the eigenspace of the lowest eigenmodes, and apply ORA on the orthogonal space in order to keep a constant accuracy as the condition number changes. As before, this changes the complexity to

$$\begin{aligned}
\mathcal{C}_{ORA} &= w'V \log\left(\frac{1}{\epsilon_{CG}}\right) \log\kappa_{eff} + w''\langle {N_D}^2\rangle V^2, \\
\mathcal{S}_{ORA} &= 8N_c(N_c+2+2N_O)V + 8N_c\langle N_D\rangle V.
\end{aligned} \tag{35}$$

where the angular brackets denote averaging over the sample of configurations used. From available data on the growth of N_O required to keep the relative error fixed with growing κ [7] it seems that it is better to increase N_O rather than to keep κ_{eff} fixed by increasing $\langle {N_D}^2\rangle$ with increasing volume at fixed physics.

In Tables V, VI and VII we show the results of our numerical tests of the ORA using the set of c_k and d_k for $N_O = 14$ from [7] for the three configurations already described. One sees that with higher precision of inversion, ϵ_{CG}, the relations (18) indeed get satisfied well. The gradual deterioration of the performance of ORA with fixed N_O in going to larger κ is clear from the tables. This indicates that the performance of ORA may improve if a degree of adaptability can be built into the algorithm by, for example, allowing for changes in N_O

because with $N_O = 14$, we cannot obtain better precision than 0.00001, 0.00002 and 0.003 for the GW relation for configurations A, B and C respectively.

V. FIXED ORDER: ZOLOTAREV APPROXIMATION

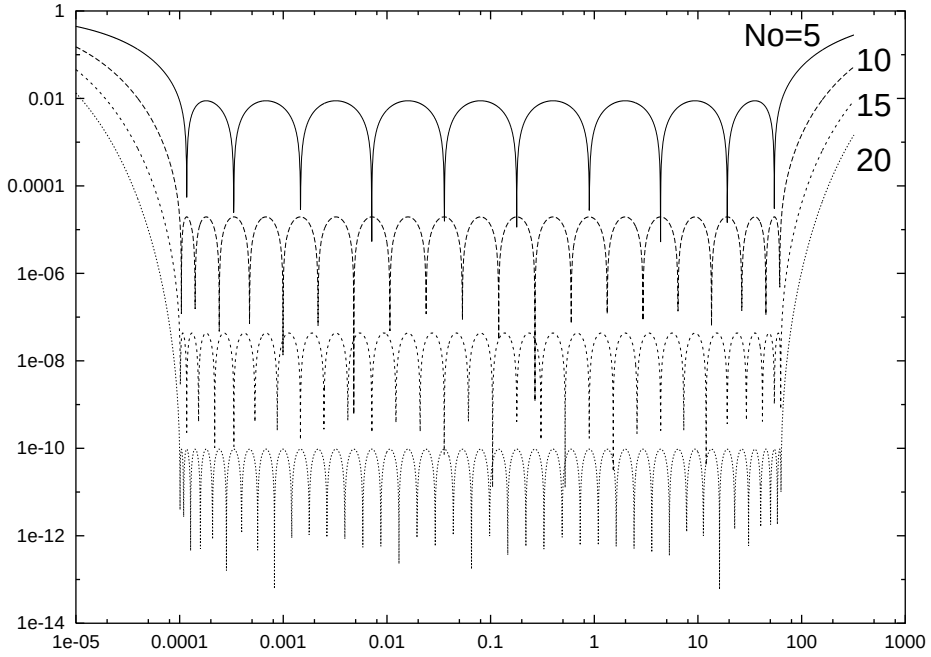

FIG. 3: The relative error, $|\epsilon(z)|\sqrt{z}$ for ZA in the range $[10^{-4}, 32]$ for computation in and outside the range for various N_O.

The Zolotarev algorithm was explored in [8] and used in [9]. It is a rational expansion defined by

$$L\Phi = \sum_{l=1}^{N_O} \left(\frac{b_l}{M + d_l} \right) \Phi, \tag{36}$$

in the range $[z_{\min}, z_{\max}]$ (the smallest and largest eigenvalues of M must satisfy the conditions $z_{\min} \leq \lambda_{\min} \leq \lambda_{\max} \leq z_{\max}$), and the expansion coefficients are

$$d_l = c_{2l-1} \qquad \text{and} \qquad b_l = d_0 \frac{\prod_{i=1}^{N_O-1}(c_{2i} - c_{2l-1})}{\prod_{i=1,i\neq l}^{N_O-1}(c_{2i-1} - c_{2l-1})}. \tag{37}$$

The c_l's are

$$c_l = \frac{\mathrm{sn}^2(lK/2N_O; \sqrt{1 - z_{\min}/z_{\max}})}{\mathrm{cn}^2(lK/2N_O; \sqrt{1 - z_{\min}/z_{\max}})}, \tag{38}$$

N_O	ϵ_{CG}	N_{CG}	$\epsilon_{1/2}$	ϵ_{GW}	ϵ_{CC}	time
5	10^{-1}	22	0.207×10^{-1}	0.281×10^{-1}	0.534×10^{-1}	0.3
6	10^{-2}	56	0.144×10^{-2}	0.216×10^{-2}	0.612×10^{-2}	0.7
7	10^{-3}	91	0.114×10^{-3}	0.160×10^{-3}	0.668×10^{-3}	1.2
8	10^{-4}	126	0.984×10^{-5}	0.150×10^{-4}	0.526×10^{-4}	1.7
9	10^{-5}	159	0.849×10^{-6}	0.123×10^{-5}	0.467×10^{-5}	2.2
10	10^{-6}	191	0.808×10^{-7}	0.117×10^{-6}	0.365×10^{-6}	2.6
11	10^{-7}	223	0.808×10^{-8}	0.119×10^{-7}	0.343×10^{-7}	3.1
13	10^{-8}	259	0.616×10^{-9}	0.924×10^{-9}	0.288×10^{-8}	3.8
14	10^{-9}	295	0.566×10^{-10}	0.798×10^{-10}	0.289×10^{-9}	4.4
15	10^{-10}	326	0.557×10^{-11}	0.793×10^{-11}	0.296×10^{-10}	5.0
16	10^{-11}	357	0.553×10^{-12}	0.723×10^{-12}	0.225×10^{-11}	5.5

TABLE VIII: Runs with the ZA in the interval $[0.035, 32]$ for configuration A. The last column gives the CPU seconds used on a Fujitsu VPP5000.

N_O	ϵ_{CG}	N_{CG}	$\epsilon_{1/2}$	ϵ_{GW}	ϵ_{CC}	time
7	10^{-1}	71	0.148×10^{-1}	0.243×10^{-1}	0.540×10^{-1}	0.9
10	10^{-2}	331	0.876×10^{-4}	0.130×10^{-3}	0.798×10^{-2}	4.5
12	10^{-3}	362	0.833×10^{-5}	0.110×10^{-4}	0.499×10^{-2}	5.2
14	10^{-4}	394	0.814×10^{-6}	0.101×10^{-5}	0.160×10^{-2}	5.9
16	10^{-5}	426	0.741×10^{-7}	0.902×10^{-7}	0.366×10^{-6}	6.6
18	10^{-6}	457	0.744×10^{-8}	0.976×10^{-8}	0.297×10^{-7}	7.3
20	10^{-7}	488	0.692×10^{-9}	0.103×10^{-8}	0.313×10^{-8}	8.1
22	10^{-8}	516	0.693×10^{-10}	0.963×10^{-10}	0.278×10^{-9}	8.9
24	10^{-9}	544	0.644×10^{-11}	0.903×10^{-11}	0.253×10^{-10}	9.7
26	10^{-10}	572	0.715×10^{-12}	0.906×10^{-12}	0.244×10^{-11}	10.4

TABLE IX: Runs with the ZA in the interval $[7.2 \times 10^{-5}, 32]$ for configuration B. The last column indicates the CPU seconds used on a Fujitsu VPP5000.

N_O	ϵ_{CG}	N_{CG}	$\epsilon_{1/2}$	ϵ_{GW}	ϵ_{CC}	time
10	10^{-1}	325	0.163×10^{-1}	0.247×10^{-1}	0.545×10^{-1}	4.5
16	10^{-2}	871	0.242×10^{-4}	0.125×10^{-2}	0.111×10^{-1}	13.4
20	10^{-3}	899	0.121×10^{-5}	0.107×10^{-5}	0.771×10^{-2}	14.9
23	10^{-4}	933	0.119×10^{-6}	0.105×10^{-6}	0.325×10^{-2}	16.3
26	10^{-5}	968	0.126×10^{-7}	0.103×10^{-7}	0.325×10^{-2}	17.7
30	10^{-6}	997	0.462×10^{-8}	0.482×10^{-9}	0.749×10^{-9}	19.4
36	10^{-7}	1024	0.521×10^{-8}	0.150×10^{-10}	0.450×10^{-10}	21.7

TABLE X: Runs with the ZA in the interval $[8.9 \times 10^{-9}, 32]$ for configuration C. The last column indicates the CPU seconds used on a Fujitsu VPP5000.

where the values of the Jacobi elliptic functions, $\mathrm{sn}(u,k) = \sin\eta$ and $\mathrm{cn}(u,k) = \cos\eta$, are defined by the integral

$$u(\sin(\eta)) = \int_0^{\sin(\eta)} \frac{dt}{\sqrt{(1-t^2)(1-k^2t^2)}}. \tag{39}$$

The constant in Eq. (38), $K = u(1)$, is the complete elliptic integral. When sn is near 0 or 1, high precision in the expressions of the coefficients of the corresponding c's is essential. The constant d_0 in Eq. (37) can be expressed in term of elliptic theta function [16], or *equivalently*, fixed by the condition [8]

$$\min_z \sum_{i=1}^{N_O} \left(\frac{\sqrt{z} b_l}{z + d_l} \right) + \max_z \sum_{i=1}^{N_O} \left(\frac{\sqrt{z} b_l}{z + d_l} \right) = 2. \tag{40}$$

As in the ORA, the multimass CG which is used to invert the terms in Eq. (36) should have a stopping criterion, ϵ_{CG}. One advantage of ZA over ORA is that the quantities d_l in Eq. (36) are larger than those in Eq. (31). As a result, a multimass CG inverter can evaluate this approximation somewhat faster. Another advantage, emphasized in [8] is that N_O required for a certain accuracy is smaller for ZA than for ORA. It was found that a relative accuracy of better than 1 part in 10^5 is obtained for the interval $[0.01, 1]$ with $N_O \approx 6$ in ZA, as compared to 14 in ORA. As shown in Figure 3, the relative error for ZA in the range $[10^{-4}, 32]$ does not require significantly higher N_O for similar control over error. However, the low adaptability, $\mathcal{A} \simeq 0.01$, means that the coefficients should be computed over a range appropriate to the condition number of the matrix. The main effect of increasing the range

for a fixed N_O is to change the coefficients b_l and d_l in such a way that the logarithmic range of d_l increases. We found that a factor 100 decrease in $z_{\min}$ (for fixed $z_{\max} = 32$) led to a factor 20–30 decrease in the ratio of the minimum and maximum values of d_l.

The complexity and spatial complexity of ZA are very similar to that of ORA. The complexity is dominated by the matrix-vector multiplication in the CG inversions, and the memory requirement is dominated by the vectors in the multimass CG. Hence

$$\begin{aligned} \mathcal{C}_{ZA} &\simeq w' V \log\left(\frac{1}{\epsilon_{CG}}\right) \log\kappa, \\ \mathcal{S}_{ZA} &= 8N_c(N_c + 2 + 2N_O)V, \end{aligned} \tag{41}$$

where w' is some constant, N_c is the number of colors and V is the lattice volume. Since the effect of deflation is also similar to that in ORA, we do not repeat that discussion here.

The performance of the Zolotarev algorithm in numerical tests is summarized in Tables VIII, IX and X. One needs to tune two algorithmic parameters, N_O and ϵ_{CG} for better efficiency. For a given value of ϵ_{CG}, we increase N_O until a saturation in the value of error is evident. For $N_O \approx 6 - 8$, the performance is similar to that of the ORA. The improvement with increasing N_O as κ increases further indicates that the ZA and ORA should both improve if N_O is allowed to change algorithmically with configuration. Such a method can be constructed from the results of [8], when N_O is increased until the maximum relative error (see Figure 3) attains a fraction ($<1/2$) of the desired accuracy. This can be implemented at the initialization step from the knowledge of the minimum and maximum of the relative error (defined in LHS of Eq.(40) or from the elliptic theta functions).

VI. ADAPTIVE ALGORITHM: CONJUGATE GRADIENT APPROXIMATION

The first adaptive method used to compute $M^{-1/2}$ was based on Lanczös algorithm [6]. In the original suggestion, the number of Lanczös steps to be taken in order to reach a given precision was investigated in terms of the variation of the eigenvalues of M with the number of Lanczös steps. A stopping criterion *à la* Conjugate Gradient was proposed but its relation to the precision was not direct. A related adaptive method based on the Conjugate Gradient algorithm was used in [11]. Here the stopping criterion is put on the residual vector in the inversion of M. This enables a direct control over the precision.

The CGA starts with an iteration which is almost the same as the usual CG algorithm

ϵ_{CG}	N_{CG}	$\epsilon_{1/2}$	ϵ_{GW}	ϵ_{CC}	time
10^{-1}	22	0.230×10^{-1}	0.410×10^{-1}	0.860×10^{-1}	0.5
10^{-2}	55	0.178×10^{-2}	0.325×10^{-2}	0.124×10^{-1}	1.2
10^{-3}	90	0.145×10^{-3}	0.279×10^{-3}	0.195×10^{-2}	2.1
10^{-4}	125	0.121×10^{-4}	0.266×10^{-4}	0.148×10^{-3}	2.9
10^{-5}	158	0.103×10^{-5}	0.219×10^{-5}	0.134×10^{-4}	3.6
10^{-6}	190	0.925×10^{-7}	0.182×10^{-6}	0.926×10^{-6}	4.2
10^{-7}	222	0.899×10^{-8}	0.172×10^{-7}	0.839×10^{-7}	5.0
10^{-8}	257	0.859×10^{-9}	0.175×10^{-8}	0.827×10^{-8}	5.7
10^{-9}	293	0.784×10^{-10}	0.158×10^{-9}	0.733×10^{-9}	6.6
10^{-10}	325	0.708×10^{-11}	0.142×10^{-10}	0.882×10^{-10}	7.3
10^{-11}	358	0.714×10^{-12}	0.149×10^{-11}	0.600×10^{-11}	8.2

TABLE XI: Runs with the CGA on configuration A ($\kappa = 10^3$). The last column indicates the CPU seconds taken on a Fujitsu VPP5000.

ϵ_{CG}	N_{CG}	$\epsilon_{1/2}$	ϵ_{GW}	ϵ_{CC}	time
10^{-1}	23	0.236×10^{-1}	0.394×10^{-1}	0.865×10^{-1}	0.5
10^{-2}	212	0.198×10^{-2}	0.347×10^{-2}	0.136×10^{-1}	4.7
10^{-3}	335	0.617×10^{-4}	0.114×10^{-3}	0.544×10^{-2}	7.4
10^{-4}	365	0.633×10^{-5}	0.112×10^{-4}	0.460×10^{-2}	8.1
10^{-5}	397	0.651×10^{-6}	0.124×10^{-5}	0.160×10^{-2}	8.8
10^{-6}	428	0.625×10^{-7}	0.117×10^{-6}	0.544×10^{-6}	9.5
10^{-7}	459	0.629×10^{-8}	0.116×10^{-7}	0.461×10^{-7}	10.2
10^{-8}	489	0.616×10^{-9}	0.110×10^{-8}	0.506×10^{-8}	10.9
10^{-9}	517	0.626×10^{-10}	0.109×10^{-9}	0.442×10^{-9}	11.6
10^{-10}	545	0.596×10^{-11}	0.995×10^{-11}	0.401×10^{-10}	12.2
10^{-11}	573	0.124×10^{-11}	0.111×10^{-11}	0.396×10^{-11}	12.9

TABLE XII: Runs with the CGA on configuration B ($\kappa = 4.4 \times 10^5$). The last column indicates the CPU seconds taken on a Fujitsu VPP5000.

ϵ_{CG}	N_{CG}	$\epsilon_{1/2}$	ϵ_{GW}	ϵ_{CC}	time
10^{-1}	62	0.231×10^{-1}	0.381×10^{-1}	0.869×10^{-1}	1.4
10^{-2}	642	0.393×10^{-2}	0.605×10^{-2}	0.153×10^{-1}	14.7
10^{-3}	830	0.310×10^{-3}	0.125×10^{-2}	0.874×10^{-2}	19.2
10^{-4}	863	0.298×10^{-4}	0.152×10^{-5}	0.691×10^{-2}	20.2
10^{-5}	891	0.287×10^{-5}	0.169×10^{-6}	0.421×10^{-2}	20.7
10^{-6}	922	0.304×10^{-6}	0.193×10^{-7}	0.325×10^{-2}	21.6
10^{-7}	957	0.318×10^{-7}	0.245×10^{-8}	0.325×10^{-2}	22.4
10^{-8}	987	0.853×10^{-8}	0.822×10^{-8}	0.231×10^{-8}	23.1
10^{-9}	1015	0.766×10^{-8}	0.135×10^{-8}	0.555×10^{-8}	23.8

TABLE XIII: Runs with the CGA on configuration C ($\kappa = 3.6 \times 10^9$). The last column indicates the CPU time in seconds on a Fujitsu VPP5000.

for the inversion of M—

1. Start from $r_1 = \Phi$, $p_1 = r_1$ and $\beta_1 = 0$,
2. Iterate as in regular CG, $\alpha_i = |r_i|^2/(p_i^\dagger M p_i)$, $r_{i+1} = r_i - \alpha_i M p_i$, $\beta_{i+1} = |r_{i+1}|^2/|r_i|^2$, and $p_{i+1} = \beta_{i+1} p_i + r_{i+1}$.
3. Stop when $|r_{i+1}| < \epsilon_{CG}$.

Note that the the only difference from the usual CG is that the vector which is $M^{-1}\Phi$ does not need to be obtained during the iteration.

In the orthonormal basis of $q_i = r_i/|r_i|$, the matrix M is the composition of the matrix Q whose i-th column is q_i, and a symmetric tridiagonal matrix, T,

$$M = Q^\dagger T Q, \quad \text{where} \quad T_{ii} = \frac{1}{\alpha_i} + \frac{\beta_i}{\alpha_{i-1}}, \quad \text{and} \quad T_{i,i+1} = -\frac{\sqrt{\beta_{i+1}}}{\alpha_i}, \tag{42}$$

where α_i and β_i are defined in the iteration above. Then compute the eigenvalues and eigenvectors of this truncated tridiagonal matrix T,

$$T = U \Lambda U^\dagger \tag{43}$$

where Λ is the diagonal matrix of the eigenvalues and U the matrix of the eigenvectors in the basis Q. The CGA solution is

$$L[\Phi] = Q^t U \Lambda^{-1/2} U^\dagger Q \Phi / |\Phi|. \tag{44}$$

The adaptability of the algorithm arises from the fact that we retain only the vectors q_i which contribute significantly to the inverse of M and we stop the iterations for $i = N_{CG}$ when $|r_{N_{CG}+1}| < \epsilon_{CG}$.

The contribution to $M^{-1/2}$ of the smallest eigenvalue $1/\lambda_{\min}$ of M will be only $1/\sqrt{\lambda_{\min}}$. Since the stopping criterion $|r_{i+1}| < \epsilon_{CG}$ is meant to compute M^{-1} it is more stringent than required. One can be more generous for $M^{-1/2}$, and use instead the stopping criterion

$$|r_{i+1}| < \epsilon_{CG}/\sqrt{\lambda_0^{(i)}} \tag{45}$$

where $\lambda_0^{(i)}$ is an upper bound of $\lambda_{\min}$. Fortunately a reasonable estimate can be obtained at each iteration i without large overheads. For any tridiagonal matrix T of order N_O, the number of eigenvalues greater than a fixed number μ is the number of positive values of $d^{(j)}$, where this set of numbers is defined by $d^{(1)} = T_{11} - \mu$ and

$$d^{(j)} = T_{jj} - \lambda_0^{(i)} - (T_{j-1,j})^2/d^{(j-1)}, \tag{46}$$

for $2 \leq j \leq N_O$ [17]. An upper bound for $\lambda_0^{(i)}$ can always be fixed by searching for a number for which at least one of $d^{(j)}$ is non-positive. This can be done by bisection, starting from the initial estimate at the first step, $\lambda_0^{(1)} = T_{11}$. While this procedure increases the complexity by order $\log N_{CG}$, the new stopping criterion in Eq. (45) has two advantages over the usual CG stopping criterion— first, N_{CG} is reduced and, second, the method becomes better adaptable since the observed $\epsilon_{1/2}$ for a given ϵ_{CG} becomes independent of λ_{min}.

Practically, to do the computation without storing the orthonormal basis Q, one makes N_{CG} iterations to get the truncated matrix T, computes the matrix U and the diagonal Λ using standard methods [12], and then repeats the N_{CG} iterations to compute the solution $L[\Phi]$. The most stringent restriction on the algorithm seems to be that one cannot use any pre-conditioning and must always start the iterations from $p_1 = r_1 = \Phi$. This algorithm has only one parameter, ϵ_{CG}. The algorithm automatically adjusts the number of iterations to achieve the specified precision irrespective of the condition number. Thus, no configuration dependent tuning of algorithmic parameters is necessary when employing the CGA for QCD applications.

The complexity of the CGA is

$$\mathcal{C}_{CGA} \simeq 2wN_{CG}V + \omega {N_{CG}}^2 \simeq 2w'V \log\left(\frac{1}{\epsilon_{CG}}\right) \log\kappa \tag{47}$$

where ω is a number independent of V. The ${N_{CG}}^2$ term comes from the handling of the tridiagonal matrix, and can be neglected since $N_{CG} \ll V$. The space complexity is the same as that of a standard CG—

$$\mathcal{S}_{CGA} \simeq 8N_c(N_c+3)V. \tag{48}$$

Since the method is adaptive, no deflation is necessary. However, deflation reduces the condition number of the matrix, and hence could improve the complexity by reducing N_{CG}. Nevertheless, for reasons that we have already discussed in connection with CA and ORA, deflation is unlikely to improve the performance at fixed physics when taking the limit of large V.

The results of our numerical tests for this algorithm are collected in Tables XI–XIII. Note that for all three test configurations there is a threshold in ϵ_{CG} above which $\epsilon_{CC} \leq 10\epsilon_{1/2}$ and below which ϵ_{CC} is roughly constant. The threshold value of ϵ_{CG} is somewhat larger than $\lambda_{\min}$ for the configuration. Similar thresholds are also seen for the ORA and ZA. This behaviour possibly reflects the existence of a large unconverged subspace in the CG iterations.

VII. COMPARING THE ALGORITHMS

In Figure 4 we have collected different measures of performance of the four algorithms we investigated in this paper, namely, the Optimized Rational Approximation (ORA) [7], the Zolotarev Approximation (ZA, which is also a rational expansion) [8, 9], the Chebychev Approximation (CA, a polynomial expansion) [10] and the Conjugate Gradient Approximation (CGA, an iterative method) [11]. Further details can be found in Tables I–XIII.

It is clear that for modest values of the condition number of M, $\kappa \leq 10^3$, the CA is the preferred algorithm. This is clear from the figure, as well as our results for the algorithmic complexities in Eqs. (28), (33), (41) and (47). However, with increasing condition numbers the performance of CA rapidly degrades. This is visible in the figure as well as in our analysis of the adaptability in Eq. (26). We have argued earlier that these drawbacks of the CA are generic to all polynomial expansions.

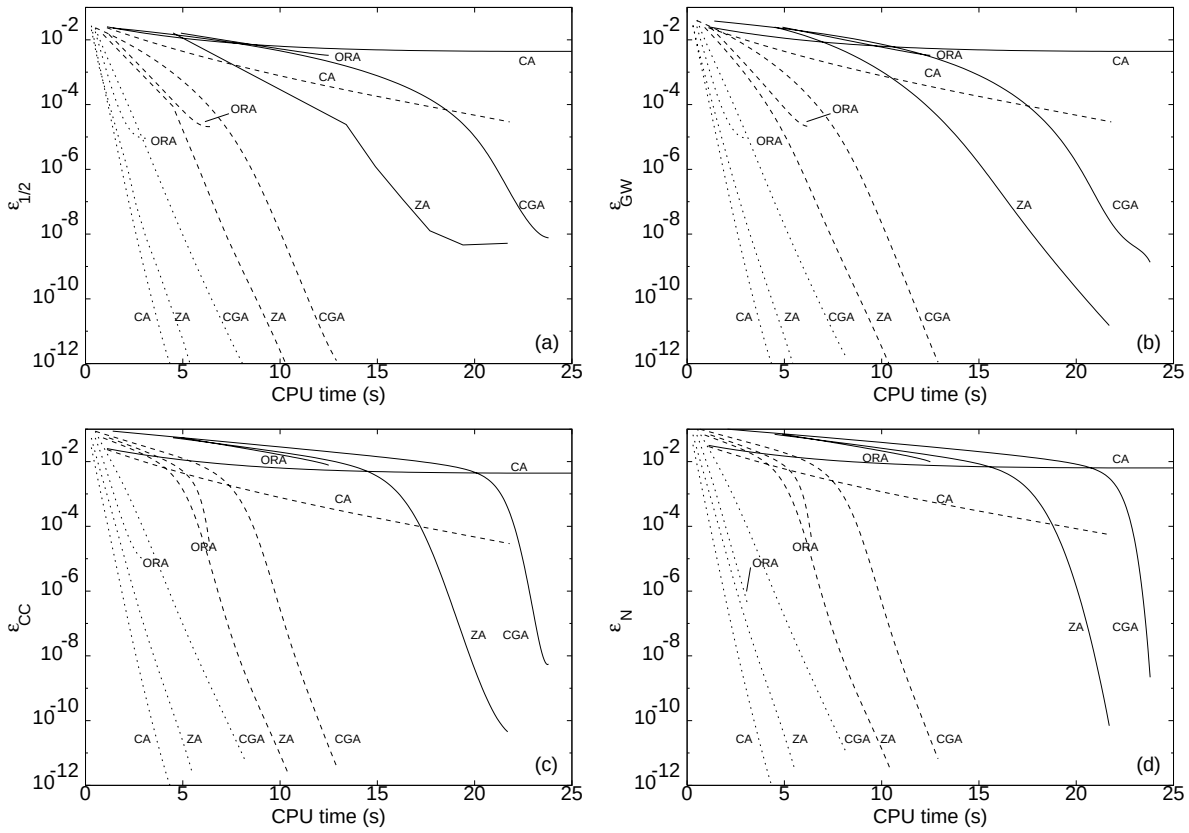

FIG. 4: Error limits as a function of the CPU time taken on a Fujitsu VPP5000— (a) $\epsilon_{1/2}$, (b) ϵ_{GW}, (c) ϵ_{CC} and (d) ϵ_N. In each case the dotted line is for configuration A ($\kappa = 10^3$), the dashed line for configuration B ($\kappa = 4.4 \times 10^5$) and the full line for configuration C ($\kappa = 3.6 \times 10^9$).

The ORA, in its present form with fixed N_O, also suffers from a lack of adaptability. In principle, this can be alleviated if the order of the approximation can be chosen adaptively. We have implemented the ZA, which is another rational approximation, for several different choices of order. As can be seen from Tables VIII- X, and from Figure 4, this improves the performance tremendously. For comparable CPU times, corresponding to low order ZA, the performance is at least one order better than that of ORA on all configurations. The key to improving the performance of rational approximations is the automatic variation of the order N_O with the condition numbers. In our tests we have simulated adaptability by working with several different orders and retained the one corresponding only to a small fraction of the inversion error. The so-tuned order is only slightly higher than that obtained in an automatic procedure defined at the end of section V.

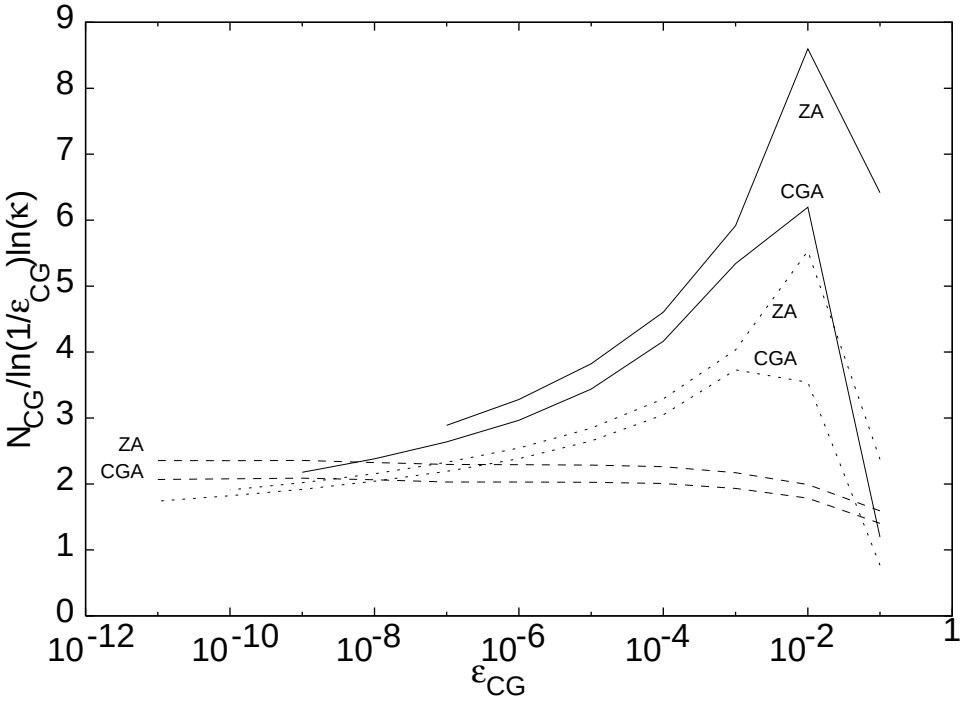

FIG. 5: The scaling of N_{CG} with κ and ϵ_{CG}. These are results of computations with configurations A (dashed lines), B (dotted lines) and C (full lines).

The CGA depends on only one parameter ϵ_{CG}. For a given value of ϵ_{CG}, the corresponding errors $\epsilon_{1/2}$ and ϵ_{GW} are almost independent of the condition number of the matrix, thanks to the relaxed stopping criterion. The price for such a good adaptability is a computing time which is 50% higher than ZA for a given accuracy (70% excess if No is small, 20% for large No). The price, however, ensures that for all the configurations one guarantees the same order of accuracy from a given value of ϵ_{CG} and with a predicted value of ϵ_{GW}.

The variation in the number of conjugate gradient iterations, N_{CG}, as the stopping criterion, ϵ_{CG}, is changed for the three configurations is shown in Figure 5. The data for the ORA are not shown in the figure because they are very similar to those of the ZA. Note that the curves for the CGA lie below that for the ZA (despite the shift in ZA as compared to CGA), which is the influence of the relaxed stopping criterion discussed in Section VI. Ref. [8] has devised a similar modification for ZA which can reduce N_{CG} in that case. Note that our above results for ZA did not use any such modifications; using it will further enhance the performance of ZA reported above.

We have noted in Section II that the relations (18) between the errors are valid for those

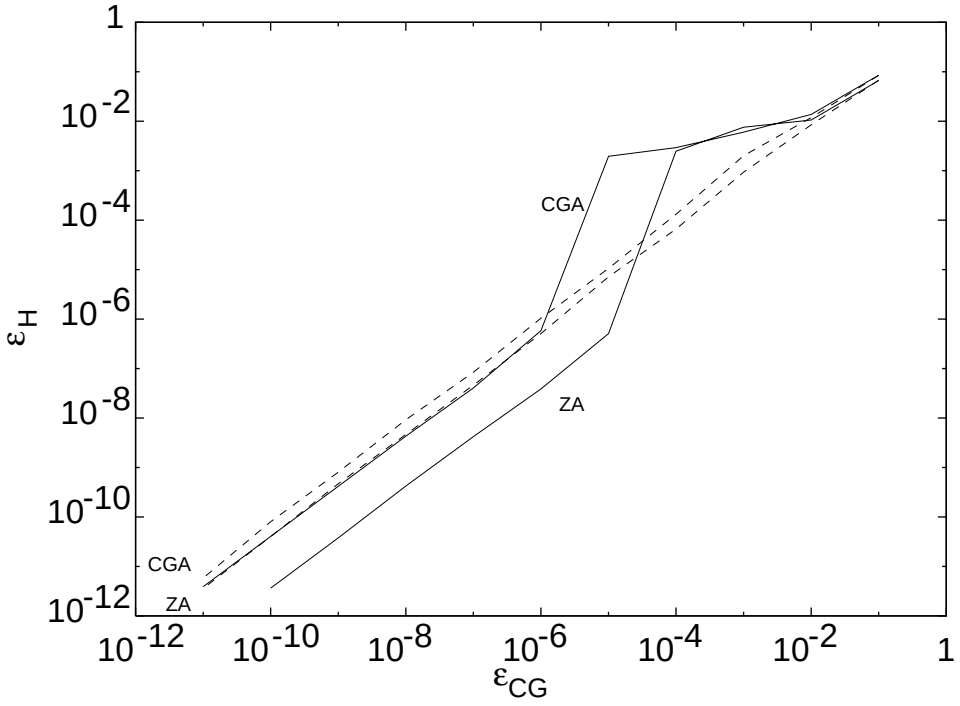

FIG. 6: The scaling of ϵ_H with ϵ_{CG} in the CGA and ZA for configurations A (dashed lines) and B (full lines).

approximations to $M^{-1/2}$ which commute with M. In particular, we noted that for the iterative algorithms these relations become valid, provided that ϵ_{CG} is sufficiently small. In Figure 6 we demonstrate this for ϵ_H, which is expected to be zero when ϵ_{CG} is small enough. For the CGA and ZA (data for the ORA are not shown because they almost coincide with that for ZA), ϵ_H decreases with ϵ_{CG}. The slopes in this plot correspond to linear decrease when ϵ_{CG} is sufficiently small. Clear non-linearities are present for larger ϵ_{CG} when the condition number is large. We believe that these non-linearities are due to large non-converged subspaces, implying a need for high accuracy.

For fixed order algorithms the adaptability, $\mathcal{A}$, quantifies the configuration dependence of speed. The numerical study can be used more directly to illustrate the adaptability by studying the slowdown in going from configuration A to B (*i.e.*, from $\kappa = 10^3$ to 4.4×10^5) or from A to C (κ changes from 10^3 to 3.6×10^9). As shown in Figure 7, both ZA and CGA are adaptable algorithms over a wide range of $\epsilon_{1/2}$. Since ZA is faster, as seen in Figure 4, it is thus the method of choice. Note, however, that CGA is very comparable to it, and may be preferred for its self-tuning ability.

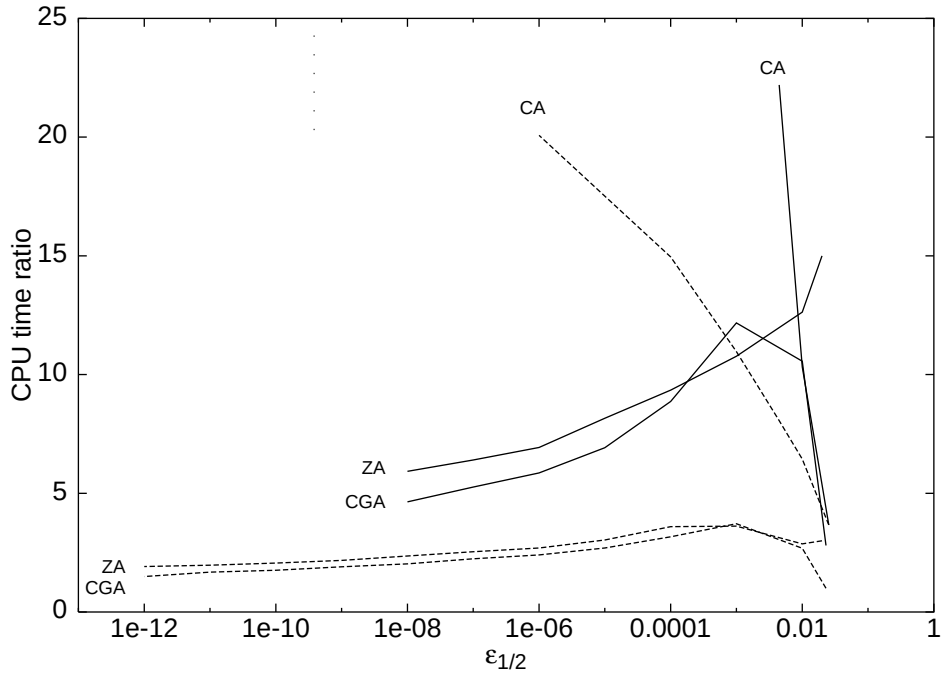

FIG. 7: The ratio of CPU time taken at fixed error, $\epsilon_{1/2}$, for (a) configurations B and A (dashed lines) and (b) configurations C and A (full lines) for three different algorithms.

We emphasise that a fair test of relative performance of algorithms is to work without deflation. First, deflation improves the performance of each of the algorithms we have investigated. Details are given in the sections on each algorithm. Nevertheless the algorithms based on rational approximation seems to be less sensitive to deflation than other ones because of the positive shifts introduced in the matrix. Second, since the computation of the eigensystem of M, necessary to deflation, is done at finite accuracy, it introduces extra errors. If the error in the computation of the eigenvalue λ_i is δ_i, then the contribution to $\epsilon_{1/2}$ is δ_i/λ_i. Thus, if we want to achieve a given $\epsilon_{1/2}$, then we must keep $|\delta_i| \leq \epsilon_{1/2}|\lambda_i|$. When the condition number κ increases, this criterion becomes impossible to satisfy, leading to catastrophic loss of accuracy.

We feel it worth pointing out that deflation is only one of many possible methods to decrease the effective condition number of the problem. Other preconditioning methods have not been seriously explored for overlap Fermions. The cost of accurate numerical methods seems to suggest that numerically stable preconditioning methods will pay a big dividend in this problem.

This work was supported by the Indo-French Centre for Promotion of Advanced Research under project number 2104-2.

---

[1] H. Neuberger, *Phys. Lett.*, B 417 (1998) 141 [hep-lat/9707022].

[2] P. H. Ginsparg and K. G. Wilson, *Phys. Rev.*, D 25 (1982) 2649.

[3] T. W. Chiu, *Phys. Rev.*, D 58 (1998) 074511 [hep-lat/9804016].

[4] H. Neuberger, *Phys. Rev. Lett.*, 81 (1998) 4060 [hep-lat/9806025].

[5] P. Hernández, K. Jansen, M. Lüscher, *Nucl. Phys.*, B 552 (1999) 363 [hep-lat/9808010].

[6] A. Boriçi, *J. Comput. Phys.*, 162 (2000) 123 [hep-lat/9910045].

[7] R. G. Edwards, U. Heller, R. Narayanan, *Nucl. Phys.*, B 540 (1999) 457 [hep-lat/9905028].

[8] J. van den Eshof *et al.*, *Computer Phys. Comm.* 146 (2002) 203 [hep-lat/0202025].

[9] T.-W. Chiu and T.-H. Hsieh, hep-lat/0204009; S. J. Dong *et al.*, hep-lat/0304005.

[10] P. Hernández, K. Jansen, L. Lellouch, *Phys. Lett.*, B 469 (1999) 198 [hep-lat/9907022].

[11] R. V. Gavai, S. Gupta, R. Lacaze, *Phys. Rev.*, D 65 (2002) 094504 [hep-lat/0107022].

[12] For a general analysis of error propagation in Gram-Schmidt orthogonalization, see G. H. Golub and C. F. van Loan, *Matrix Computations*, John Hopkins University Press, 1996.

[13] R. G. Edwards, U. Heller, R. Narayanan, *Phys. Rev.*, D 61 (2000) 074504 [hep-lat/9910041].

[14] S. J. Dong *et al.*, *Phys. Rev. Lett.*, 85 (2000) 5051 [hep-lat/0006004].

[15] T. DeGrand, *Phys. Rev.*, D 63 (2001) 034503 [hep-lat/0007046].

[16] T. W. Chiu *et al.*, hep-lat/0206007.

[17] See, section 11.4 in P. Lascaux and R. Théodor, *Analyse numérique matricielle appliqueé à l'art de l'ingénieur, (2)*, Masson, Paris, 1994; see also section 8.5.2 in [12].

[18] See, for example, the section on matrix square roots in R. A. Horn and C. R. Johnson, *Topics in Matrix Analysis*, Cambridge University Press, 1991.